UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/620,714	07/20/2000	Luke Matthew Browning	AUS9-2000-0277-US1	3354

35525 7590 08/26/2004

IBM CORP (YA)
C/O YEE & ASSOCIATES PC
P.O. BOX 802333
DALLAS, TX 75380

EXAMINER	
STEELMAN, MARY J	
ART UNIT	**PAPER NUMBER**
2122	

DATE MAILED: 08/26/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
UNITED STATES PATENT AND TRADEMARK OFFICE
P.O. BOX 1450
ALEXANDRIA, VA 22313-1450
www.uspto.gov

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application Number: 09/620,714
Filing Date: July 20, 2000
Appellant(s): BROWNING ET AL.

Duke W. Yee, Reg. No. 34285

For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed 19 April 2004.

(1) Real Party in Interest

A statement identifying the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

A statement identifying the related appeals and interferences which will directly affect or be directly affected by or have a bearing on the decision in the pending appeal is contained in the brief.

(3) Status of Claims

The statement of the status of the claims contained in the brief is correct.

(4) Status of Amendments After Final

No amendment after final has been filed.

(5) Summary of Invention

The summary of invention contained in the brief is correct.

(6) Issues

The appellant's statement of the issues in the brief is correct.

(7) Grouping of Claims

Appellant's brief includes a statement that claims 1-29 do not stand or fall together and provides reasons as set forth in 37 CFR 1.192(c)(7) and (c)(8).

(8) Claims Appealed

The copy of the appealed claims contained in the Appendix to the brief is correct.

(9) Prior Art of Record

5,560,009	LENKOV ET AL.	9-1996
6,240,529	KATO	5-2001
6,412,106	LEASK ET AL.	6-2002

(10) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

Claims 1-6, 11-18, and 23-29 are rejected under 35 U.S.C. 102(e) as being anticipated by Kato. Claims 7-9 and 19-21 are rejected under 35 U.S.C 103(a) as being unpatentable over Kato in view of Lenkov. Claims 10 and 22 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kato in view of Leask et al.

(11) Response to Argument

Appellant has filed a "REQUEST FOR REINSTATEMENT OF APPEAL UNDER 37 CFR 1.193(B)(2)". Page 2, 2nd paragraph recites, "...Examiner has improperly reopened prosecution of this case in the most recent Office Action dated 03/09/2004. Per 1208.01 (entitled "prohibition Against Entry of New Ground of Rejection in Examiner's Answer")..." In response, Examiner would like to point out that the Office Action dated 03/09/2004 was a withdrawal of the previous Final Office Action (07/30/03) for the purpose of clarifying issues that appeared to be unclear to Applicant, as presented in the Appellant's Brief, filed 12/24/03. The Office Action was not an Examiner's Answer, but rather a new Final Office Action, as necessitated by the last amendments filed 06/02/2003. It should be noted that throughout the entire prosecution of this case, the prior art references have remained unchanged. New grounds of rejection have not been applied.

(A) Appellant's arguments regarding claims 1-5, 7-17, and 19-25 (**Group I**):

I. On page 6, 4th paragraph, of Supplemental Appellant's Brief, dated 19 April 2004; Appellant asserts that "Kato's restoration is responsive to a state restoration command *currently* being issued. This does not teach or otherwise disclose that claimed step of retrieving the stored process state *in response to a predefined event.* The cited reference requires a *manual restoration* of a state storage file as selected by a user from a list of existing files...this retrieval is *responsive to a user input command.*" (Emphasis added by Appellant.)

II. Additionally, on page 8, 2nd paragraph, Appellant asserts, "...the teachings of the Kato reference are directed to *assisting the user in manually selecting* from one of many state storage files to be used when the *user manually issues a state restore command*" which "cannot reasonably be construed to read on the claimed feature of retrieving the stored process state *in response to a predefined event.*" (Emphasis added by Appellant.)

Examiners Response to the Appellant's arguments is as follows:

I. First, Kato indeed teaches the "predefined event", the claimed step of "retrieving the stored process state in response to a predefined event" at col. 8, lines 29-35, which states, "When a state restoration command (a predefined event) is issued.... state restoration unit reads in the file and restores the debugged state." (emphasis added)

II. Second, note that the plain language of the claim does not exclude and / or include automatic or manual user intervention in the restoration of state information. Thus there is no need to read 'automatic' as Appellant asserts.

(B) Appellant's arguments regarding claims 6 and 18 (**Group II**):

I. As noted on page 9, 2nd paragraph, "...the cited reference does not teach the claimed step of 'reinitiating debugging from the stored process state, wherein the process has control over at least one child process and the process state includes a process descriptor for each of the at least one child process." Kato does not teach "wherein the process has control over at least one child process and the process state includes a process descriptor for each of the at least one child process." Appellant claims (page 10, 1st paragraph) that invention "provides process state information for both (1) the process and (2) the at least one child process."

II. Additionally, on page 10, 1st paragraph, Appellant states, "The cited reference only alludes to maintaining debug capabilities for a single running process." Whereas, Appellant's invention provides "debug capabilities in an environment with multiple concurrently running processes."

Examiners Response to the Appellant's arguments is as follows:

I. It should be noted that Appellant's argument asserting the invention "provides process state information for both (1) the process and (2) the at least one child process" is not in the claim language of either claim 6, or claim 18.

Kato disclosed the limitations of claim 6 ("...reinitiating debugging from the stored process state, wherein the process has control over at least one child process..." at

col. 8, lines 16-19, as such "a function to manage a situation upon storage of a debugged state is further added (a child process). In this connection, storage situation management unit is additionally provided for state storage unit." See FIG. 5A, (col. 6, lines 51-52) "a flow chart illustrating a processing flow (a process) of a debugging method..." (emphasis added) Also note, col. 6, lines 30-34, "...the means for managing a situation when the debugged state is stored, a state storage file with which an intended debugged state can be restored can be retrieved by referring to a list or trace information (See FIG. 4), and restoration processing of a debugged state is facilitated (another child process)." (emphasis added) The debug process (FIG. 5A) calls many child processes. At FIG. 5A, #304->B, processes called include (See FIG. 5B): #322 – 'BACK EXECUTE ONE INSTRUCTION', one by one until a desired point is reached, #322 & #333 (NOTE: FIG. 5B #322 is incorrect, should be 332) – Select from a state storage file list for a checkpointed state and restore a debugged state.

-"...and the process state includes a process descriptor Process Descriptor" (col. 10, lines 1-3), "...current debug information is stored into a file (process descriptor) and is registered simultaneously into storage situation management file. In calling child processes, the debug process knows the "process descriptor" for each of the at least one child processes.

II. Again, it should be also noted that Appellant argues for such a limitation, "debugging a process in a multi-process environment". This is not in the claim language of either claim 6 or 18. The argument is moot and not persuasive. Moreover, it also

should be noted that the plain claim language merely called for "...at least one child process...for each of the at least one child process..." (emphasis added)

(C) Appellant's arguments regarding claims 26 and 27 (**Group III**):

I. As noted on page 10, 3rd paragraph, regarding claim 26 and dependent claim 27, 'the debugger may advantageously be instructed to automatically run between a checkpoint and a breakpoint repeatedly."

II. As noted on page 10, 3rd paragraph, regarding claim 26 and dependent claim 27, "...using a plurality of register or memory variable values..."

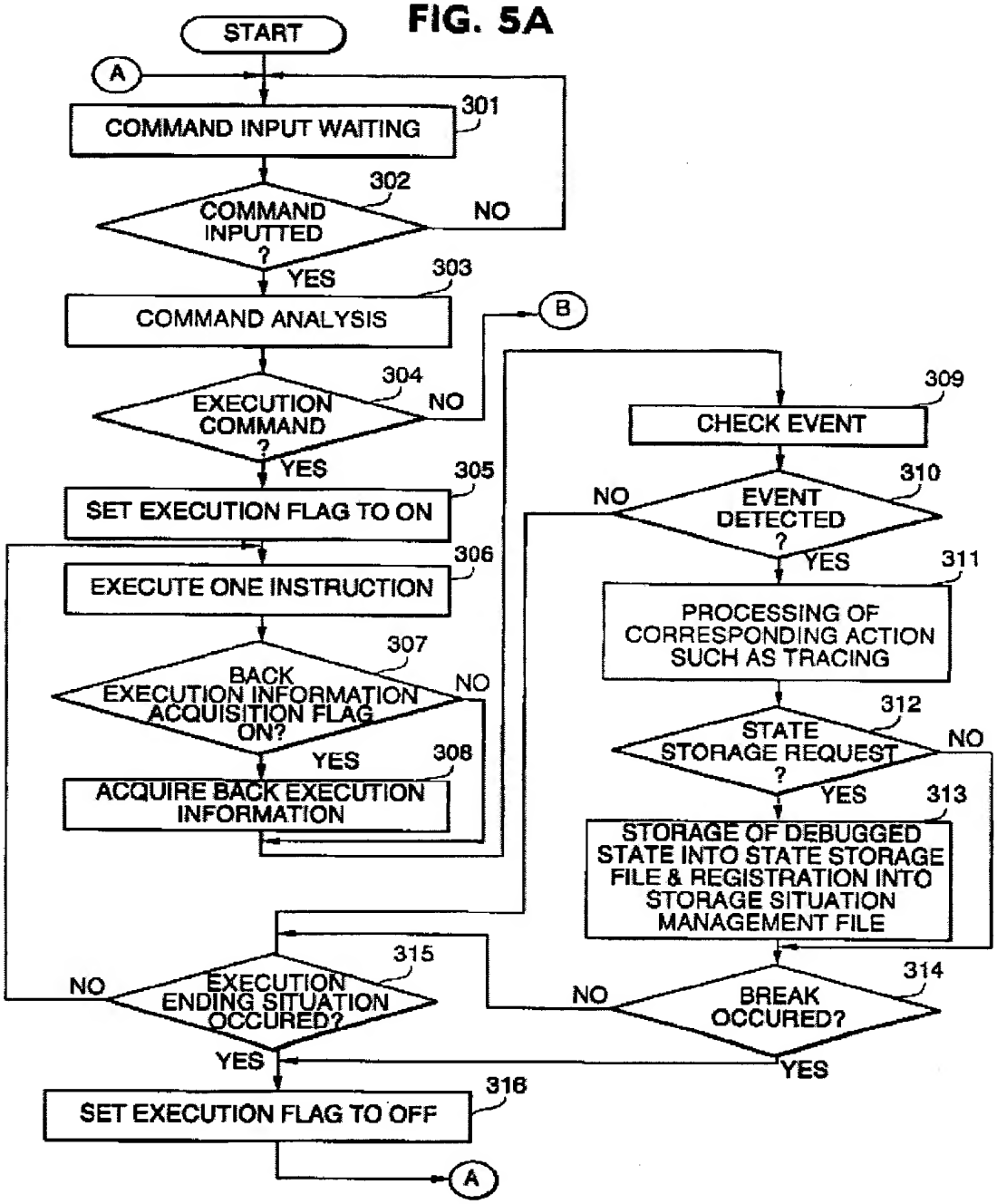
III. Additionally, on page 10, 4th paragraph, lines 23-30, Appellant asserts, "The cited reference does not teach the claimed steps of "retrieving the stored process state in response to a predefined event"... "wherein the predefined event is a checkpoint"

IV. Additionally, on page 10, 4th paragraph , in reference to Examiner's citation of Kato's FIG. 5A, #309 and #314, Appellant asserts that Kato's invention "checks whether an event has occurred and if so, processing of the action registered in connection with the event is performed", but fails to mention the "specific action of *retrieval of a stored process state in response to this event checking*." (Emphasis added by Appellant)

Examiners Response to the Appellant's arguments is as follows:

I. Appellant has asserted on page 10, lines 19-20, "...the debugger may advantageously be instructed to automatically run between a checkpoint and a breakpoint repeatedly." The word "automatically" (page 10, line 20 of Appeal Brief) is not in the claim language. To the contrary, claim 26 recites, "...repeatedly running between the checkpoint and the breakpoint for a plurality of times." Kato discloses this feature in FIG. 5A and FIG. 5B. It should be noted that Kato teaches such limitations as "repeatedly" deemed to be inherent in the "closed-loop" 'A' (from #316, SET EXECUTION FLAG TO OFF, back to #301, COMMAND INPUT WAITING). At #314, BREAK OCCURRED?, of FIG. 5A, it is determined whether there exists a request for a break (first event is a breakpoint). YES provides for cycling back via closed-loop 'A' to #304, EXECUTION COMMAND? If the result is set to 'NO', a break point is indicated, and the 'B' option is followed (to FIG. 5B). In FIG. 5B, at #329, STATE STORAGE COMMAND?, an event (predefined event is a checkpoint) is checked as to whether to store a checkpoint file or retrieve a checkpoint file. The "YES" result leads to #330 where a checkpoint file is stored. Alternately, a "NO" result leads to #331, STATE RESTORATION COMMAND? Exiting #331, following the "YES" option to #322 and #333, provides for a checkpointed file to be selected (See FIG. 4). State is restored to the debug process using the information in the checkpoint file. Execution continues/repeats to 'A' of FIG. 5A.

FIG. 5A


START
A
COMMAND INPUT WAITING
301
COMMAND INPUTTED ?
302
NO
YES
COMMAND ANALYSIS
303
B
EXECUTION COMMAND ?
304
NO
YES
SET EXECUTION FLAG TO ON
305
EXECUTE ONE INSTRUCTION
306
BACK EXECUTION INFORMATION ACQUISITION FLAG ON?
307
NO
YES
ACQUIRE BACK EXECUTION INFORMATION
308
CHECK EVENT
309
EVENT DETECTED ?
310
NO
YES
PROCESSING OF CORRESPONDING ACTION SUCH AS TRACING
311
STATE STORAGE REQUEST ?
312
NO
YES
STORAGE OF DEBUGGED STATE INTO STATE STORAGE FILE & REGISTRATION INTO STORAGE SITUATION MANAGEMENT FILE
313
EXECUTION ENDING SITUATION OCCURED?
315
NO
YES
BREAK OCCURED?
314
NO
YES
SET EXECUTION FLAG TO OFF
316
A

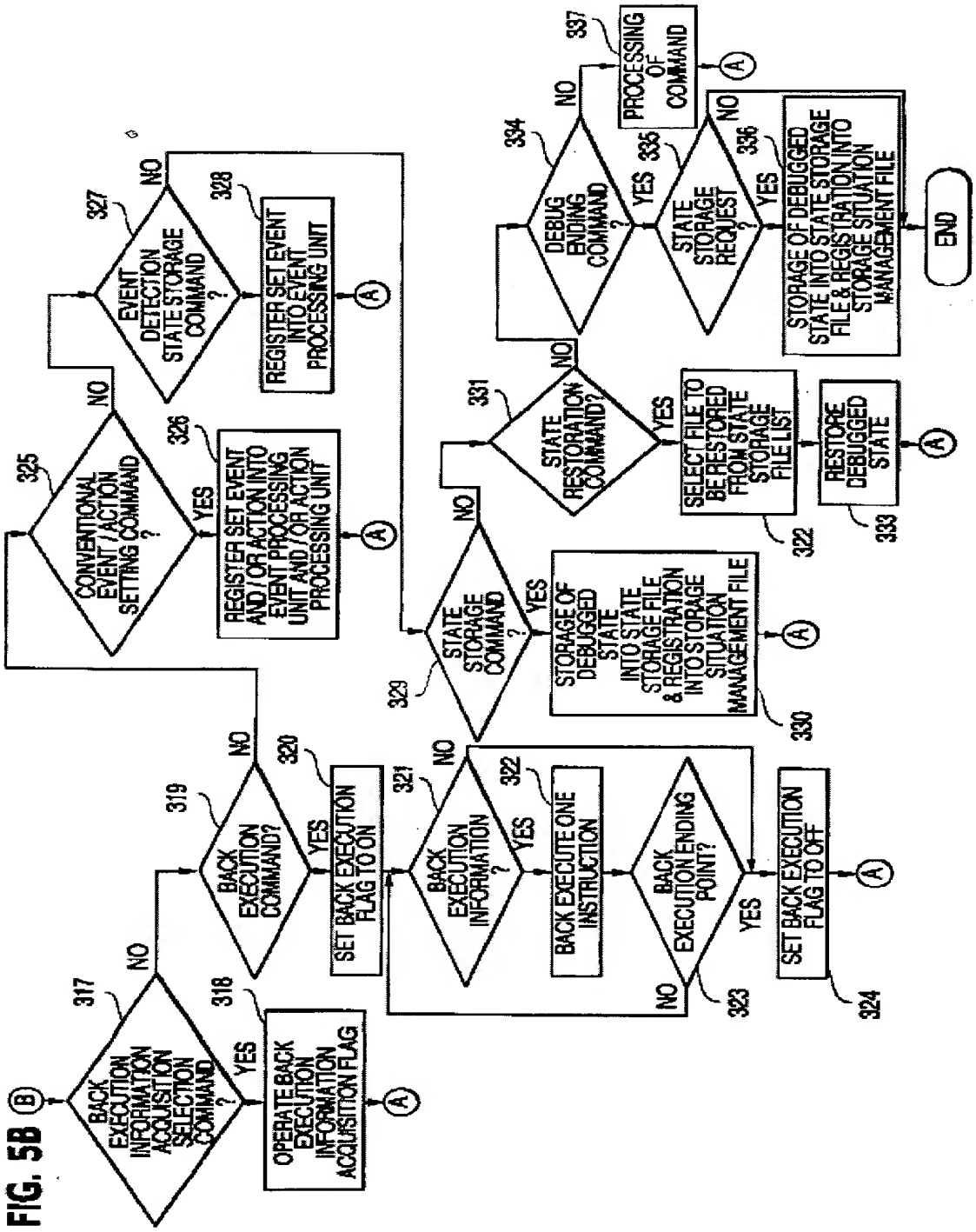

B
BACK EXECUTION INFORMATION ACQUISITION SELECTION COMMAND ?
317
NO
YES
318
OPERATE BACK EXECUTION INFORMATION ACQUISITION FLAG
A
BACK EXECUTION COMMAND?
319
NO
YES
320
SET BACK EXECUTION FLAG TO ON
BACK EXECUTION INFORMATION ?
321
NO
YES
322
BACK EXECUTE ONE INSTRUCTION
BACK EXECUTION ENDING POINT?
323
NO
YES
SET BACK EXECUTION FLAG TO OFF
324
CONVENTIONAL EVENT / ACTION SETTING COMMAND ?
325
NO
YES
326
REGISTER SET EVENT AND / OR ACTION INTO EVENT PROCESSING UNIT AND / OR ACTION PROCESSING UNIT
EVENT DETECTION STATE STORAGE COMMAND ?
327
NO
328
REGISTER SET EVENT INTO EVENT PROCESSING UNIT
STATE STORAGE COMMAND ?
329
NO
YES
STORAGE OF DEBUGGED STATE INTO STATE STORAGE FILE & REGISTRATION INTO STORAGE SITUATION MANAGEMENT FILE
330
STATE RESTORATION COMMAND?
331
NO
YES
SELECT FILE TO BE RESTORED FROM STATE STORAGE FILE LIST
322
RESTORE DEBUGGED STATE
333
DEBUG ENDING COMMAND ?
334
NO
YES
STATE STORAGE REQUEST ?
335
NO
YES
336
STORAGE OF DEBUGGED STATE INTO STATE STORAGE FILE & REGISTRATION INTO STORAGE SITUATION MANAGEMENT FILE
END
PROCESSING OF COMMAND
337
A

FIG. 5B

II. The claim language states, "variable values", not "using a plurality of register or memory variable values to assist in debugging the process (terms used in the Appeal Brief, page 10, last line of paragraph 3)." Kato disclosed (col. 8, lines 33-35), "If a file to be restored is selected from the information, the state restoration unit reads in the file and restores the debugged state." (Variable values are automatically modified as the restoration unit reads in the file to reset the state of the process.)

III. Kato disclosed the "specific action of retrieval of a stored process state in response to this event checking" at (See FIG. 5B) col. 10, lines 3-9, "If a state restoration command is inputted (331) (in response to predefined event), a file to be restored is selected (#332) from within the storage file list (FIG. 4) based on the storage situation management file which manages situations upon storage of a debugged state...and the selected file is read in to restore (retrieval of a stored process state / variable values are modified) the debugged state." (emphasis added)

IV. Appellant asserts that Kato's invention "checks whether an event has occurred and if so, processing of the action registered in connection with the event is performed", but fails to mention the "specific action of retrieval of a stored process state in response to this event checking." However, this is not in the claim language. See claims 26 and 27. Furthermore, note at #333 of FIG. 5B that the stored process state is used to restore the debug process to a checkpointed state.

(D) Appellant's arguments regarding claim 28 (**Group IV**):

As cited on page 12, last paragraph, of Supplemental Appellant's Brief, Appellant asserts, "reference does not teach the claimed steps of 'creating a child process from the debug process'", "saving a process state of the child process" (the child process having been created from a debug process) and "executing the child process using the stored process state"."

Additionally, page 13, line 3, recites, "...two processes – a debug process and a child process (which is created from the debug process)."

Examiners Response to the Appellant's argument is as follows:

Claim language fails to clarify a (1) parent debug process and (2) a child debug process created from the parent debug process. A relationship is not noted. Examiner reads the claim language to state only that a child process (of some type) is created. Claim language fails to indicate, "that the child process (created from the debug process) is the object of debugging."

(E) Appellant's arguments regarding claim 29 (**Group V**):

I. As noted on page 14, 4th paragraph, of Supplemental Appellant's Brief, "The cited reference does not teach or otherwise disclose saving a process state of a traced process and a process state of another process that is not being traced."

II. Additionally, page 15, 3rd paragraph, Appellant notes Kato, "col. 7, lines 65-66 merely describes an ability to store a debugged state into a file if a certain event

occurs". "It does not teach or otherwise disclose "saving a process state *of the traced process* and a process state of *another process that is not being traced by the debugger*." (Emphasis added by Appellant.) At best, it teaches storing a single process state, whereas the claim is directed to storing two different process states from two different processes – with one state being traced and the other one not being traced."

Examiners Response to the Appellant's arguments is as follows:

I. Examiner's cited reference: -saving a process state of the traced process and a process state of another process that is not being traced by the debugger; (Col. 7, lines 35-47 and 65-66, "...an instruction execution situation, a memory access situation and so forth at the point of time are recorded...") State is saved when 'situations' are recorded. Kato disclosed Embodiment 1 (FIG. 4), saving state without tracing, and Embodiment 2 (FIG. 6), saving state using tracing. Tracing is disclosed by Kato in Embodiment 2 of the invention. See col. 10, lines 28-41.

II. See Kato, col. 10, lines 28-41, "...while a debugged state can be stored at the timing of detection of an event, also setting such that trace information is acquired upon detection of the same event is possible." (save state of traced process, save state of a process not traced) Also, col. 8, lines 29-32, "When a state restoration command is issued, storage situation management unit displays a list of currently existing state storage files based on storage situation management file. FIG. 4 shows an example of the list." Multiple process states are saved, some may be from Embodiment 1 (not traced) and some from Embodiment 2 (traced).

In conclusion:

Claim language does not require that the checkpoint / breakpoint cycle occurs *automatically*, without user intervention. (Re: Group I, claims 1-5, 7-17, and 19-25.)

Claim language does not require "multiple concurrently running processes." (Re: Group II, claims 6 and 18.)

Claim language does not clearly require a parent debug process and it's related child debug process. (Re: Group IV, claim 28.)

Claim Rejections - 35 USC § 102

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

6. **Claims 1-6, 11-18, & 23-29** are rejected under 35 U.S.C. 102(e) as being anticipated by U.S. Patent 6,240,529 to Kato.

Per claims 1, 13, & 25:

Kato disclosed a method, apparatus (col. 6, line 11) and a program recorded on a recording medium (col. 7, line 5) for debugging a process and storing state information.

-initiating debugging of the process; (Col. 7, line 15, "When debugging is started..."); -saving a process state in response to a first event to form a stored process state; (Col. 7, lines 65-66, "storing a debugged state into a file if a certain event occurs." (a first event)); -retrieving the stored process state in response to a predefined event; (Col. 8, lines 29-35, "When a state restoration command is issued, (in response to predefined event at FIG. 5A, #304: 'NO' option leads to FIG. 5B) storage situation management unit displays a list of currently existing state storage files based on storage situation management file. FIG. 4 shows an example of the list. If a file to be restored is selected from the information, then state restoration unit reads in the file (retrieves – FIG. 5B, #333).)

-reinitiating debugging from the stored process state. (Col. 8, lines 33-35, "If a file to be restored is selected from the information, then state restoration unit reads in the file (retrieves) and restores (restore at #333, FIG 5B / reinitiates debugging at 'A', exiting from #333 and proceeding to FIG 5A) debugged state." (emphasis added) See FIG. 3, Execution units #108 & #109 (processor) and storage file, #118 (memory). FIG. 4, Execution upon restoration using saved / checkpointed state. FIG. 5B, #333.)

Per claims 2 and 14:

-first event occurs periodically. (Col. 6, lines 14-19, "storage...for...debugged state...can be designated at an arbitrary point of time...")

Per claims 3 and 15:

-process state is saved in a checkpoint data structure. (Col. 10, lines 30-32, "debugged state is stored, a state storage file...")

Per claims 4 and 16:

-checkpoint data structure is a checkpoint file. (Col. 10, lines 30-32, "debugged state is stored, a state storage file... (checkpoint data structure / checkpoint file)")

Per claims 5 and 17:

-checkpoint data structure includes a process descriptor for the process. (FIGS. 4 & 6, State Storage File Name, and col. 5, lines 43-45, "storing a state storage file name and the situation upon the storage in a correlated condition into a storage situation management file...")

Per claims 6 and 18:

-initiating debugging of the process; (Col. 7, line 15, "When debugging is started...")
-saving a process state in response to a first event to form a stored process state; (Col. 7, lines 65-66, "storing a debugged state into a file if a certain event occurs." Also see FIG. 5A, #312. If it is requested to store state, then at #313 state is stored into a file at FIG 4.);
-retrieving the stored process state in response to a second event; (See FIG. 5A. At #304 option 'B' can lead to FIG. 5B and the retrieval of a stored process state at #333.Col. 8, lines 29-35, "When a state restoration command is issued, storage situation management unit displays a list of currently existing state storage files based on storage situation management file. FIG. 4 shows an example of the list. If a file to be restored is selected from the information, then state restoration unit reads in the file (retrieves) and restores (reinitiates) the debugged state." FIG. 3, Execution units #108 & #109 (processor) and storage file, #118 (memory). FIG. 4, Execution upon restoration. FIG. 5B, #333.)

-reinitiating debugging from the stored process state, wherein the process has control over at least one child process and the process state includes a process descriptor for each of the at least one child process. (FIG. 5A. Note 'A' cycles from the 'end' of the FIG. 5A back to the 'beginning' (reinitiating). Kato disclosed (col. 8, lines 16-19), "a function to manage a situation upon storage of a debugged state is further added. (child process) In this connection, storage situation management unit is additionally provided for state storage unit. (checkpoint)" Also note, col. 6, lines 30-34, "...the means for managing a situation when the debugged state is stored (checkpoint), a state storage file with which an intended debugged state can be restored can be retrieved by referring to a list or trace information, and restoration processing of a debugged state is facilitated (another child process)."

- and the process state includes a process descriptor for each of the at least one child process. (Col. 10, lines 1-3), "...current debug information is stored into a file (process descriptor) and is registered simultaneously into storage situation management file. In calling child processes, the debug process knows the "process descriptor" for each of the at least one child processes.

Per claims 11 and 23:

-the process state is saved when the program is in a stopped state. (Col. 7, lines 65-66, "storing a debugged state into a file if a certain event occurs.")

Per claims 12 and 24:

-the stopped state is at a breakpoint. (Col. 7, lines 37-38, "A break point at which execution of a program is interrupted..." Also, col. 10, lines 20-26.)

Per claim 26:

-the first event is a breakpoint and the predefined event is a checkpoint;

(See FIG. 5A, #304 (predefined event / checkpoint), 'B', and #314, and Col. 9, lines 15-36, "Then, it is checked whether or not a state corresponding to one of events registered (checkpoint) by the user is satisfied...If a request for a break (breakpoint) is generated in response to the detection of the event (314), then the execution flag is changed to OFF (316)." (first event is a breakpoint) After cycling back via 'A' to #304, if the execution flag is set to OFF, a break point may be indicated, at which point the 'B' option is followed (to FIG. 5B). In FIG. 5B at #330 a checkpoint file is saved or at #322 AND #333, a checkpointed file is selected (See FIG. 4) and state is restored to the debug process using the information in the checkpoint file. Execution continues to 'A' of FIG. 5A. Checkpoint at FIG. 5A, #304 leads to restoring state at FIG. 5B, #333.)

-and further comprising the step of repeatedly running between the checkpoint (event registered) and the breakpoint for a plurality of times. (Execution continues from RESTORE DEBUGGED STATE at FIG. 5B, #333, to 'A' of FIG. 5A. Checkpoint at FIG. 5A, #304 leads to restoring state at FIG. 5B, #333. The 'closed loop' returns to 'A' of FIG. 5A to repeat between checkpoint and breakpoint, thereby addressing the limitation of "repeatedly running between the checkpoint and the breakpoint for a plurality of times".)

Per claim 27:

-variable values are automatically modified after retrieving the stored process state for the checkpoint. (Col. 8, lines 33-35, "If a file to be restored is selected from the information, then state restoration unit reads in the file (modify values) and restores the debugged state." Also see FIG. 3, #117, state restoration unit.)

Per claim 28:

-initiating a debug process; (Col. 7, lines 15.)

-creating a child process from the debug process; (Col. 10, lines 43-46, "...the storage situation management unit...additionally has a function of recording a debugged state storage timing into trace information." Note FIG. 5B. Seven processes are forked off of FIG, 5A, #304)

-saving a process state of the child process in response to a first event, to form a stored process state; (Col. 10, lines 49-51, "...information of the state storage file name and so forth is stored also into a buffer (saving state of child process) for trace information storage." Also see FIG. 6)

-retrieving the stored process state in response to a second event; executing the child process using the stored process state (See FIG. 5B, #333 & 'A' for re-executing). (Col. 11, lines 19-29, "...the debugged state storage file name is recorded into 407 (FIG. 6) at a timing at which the debugged state is stored...the retrieval means which refers to trace information is additionally provided, and specification of a debugged information file to be restored is facilitated by confirming a state storage file, a type of an execution instruction upon production of the file, a state of memory and so forth.")

Per claim 29:

-tracing a process by a debugger; (Col. 7, lines 35-37, "Debugger unit in debugging apparatus performs processing regarding a debugging function for breaking, tracing or the like.")

-saving a process state of the traced process and a process state of another process that is not being traced by the debugger; (Col. 7, lines 35-47 and 65-66, "...an instruction

execution situation, a memory access situation and so forth at the point of time are recorded..." Tracing is disclosed by Kato in Embodiment 2 of the invention. See col. 10, lines 33-36. Multiple process states are saved, some may be from Embodiment 1 (FIG. 4, not traced) and some from Embodiment 2 (FIG. 6, traced).)

-retrieving the saved process states; (Col. 8, lines 29-35, "When a restoration command is issued...")

-reinitiating debugging of the process using the retrieved process states. (Col. 8, lines 29-35, "When a state restoration command is issued, storage situation management unit displays a list of currently existing state storage files based on storage situation management file. FIG. 4 shows an example of the list. If a file to be restored is selected from the information, then state restoration unit reads in the file and restores the debugged state." Also col. 10, lines 33-36, "...an example wherein more precise retrieval of a debugged state to be restored is allowed by combining them with another debugging function." Also, FIGS. 5A and 5B, reinitiate debugging at 'A'.)

Claim Rejections - 35 USC § 103

7. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the
> manner in which the invention was made.

8. **Claims 7-9 and 19-21** are rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent 6,240,529 to Kato, and further in view of U.S. Patent 5,560,009 to Lenkov et al.

Kato disclosed an invention to debug and save state to a file, allowing resumption. Kato failed to provide information on data type descriptors, instance descriptors, and data block corresponding to an instance descriptor.

However, Lenkov described an invention that generated symbolic debug information, including creating a debug data structure (col. 6 , line 62), and (col. 7, lines 33-48) including various information...debug name and type tables, a table of name strings, and object file symbol table. Col. 8, lines 39-46, "The core object file class is used to read common object file data structures...includes access routines to look up symbols in symbol tables... (Col. 16, lines 57-67), "the dtab class includes member for holding state during iterations...The classes, data structures, variables and functions which comprise the dtab... (Col. 17, lines 55-60) blocktab is a derived class of dtab. Blocktab is the base class of all source blocks... (Col. 25, lines 46-51) The output object file interface creates a new object/debug file and stores...debug information." Also, (Col. 26, lines 20-21) "output object file interface is implemented in the form of classes...instantiations..." Also, (Col. 26, lines 56-58) "the preprocessor creates a source file descriptor table, a procedure descriptor table, a class descriptor table..."

Therefore, it would have been obvious, to one of ordinary skill in the art, at the time of the invention, to modify Kato's invention to debug, store state information, and restore state, by including Lenkov's invention that processed debug information, because

it arranges symbolic debug information for storage and retrieval in a meaningful manner for efficient diagnostics..

9. **Claims 10 and 22** are rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent 6,240,529 to Kato, and further in view of U.S. Patent 6,412,106 to Leask et al..

Kato disclosed an invention to debug and save state to a file, allowing resumption. Kato failed to provide information regarding modifications prior to resuming a debug process.

However Leask disclosed a debugging invention that allows for modifying values while the program is suspended.

Therefore, it would have been obvious, to one of ordinary skill in the art, at the time of the invention, to modify Kato's invention to debug and store state, by including the features in Leask's invention that allow for modifying values because it would create a quicker interactive debug session, saving time by not requiring a start from the beginning of a program.

Per claims 10 and 22:

-modifying at least one register or memory variable before resuming debugging from the stored process state. (Leask, col. 12, lines 55-56, "capability to modify values stored in variables, while the application program is suspended.")

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

APPEAL PANEL

August 18, 2004

Conferees

Mary Steelman, Examiner

Tuan Q. Dam, Supervisory Patent Examiner

Kakali Chaki, Supervisory Patent Examiner

Duke W. Yee, Reg. No. 34,285

IBM (YA)

C/O YEE & ASSOCIATES, P.C.

P.O. Box 802333

Dallas, Texas 75380